# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2004-013190
(43)Date of publication of application : 15.01.2004

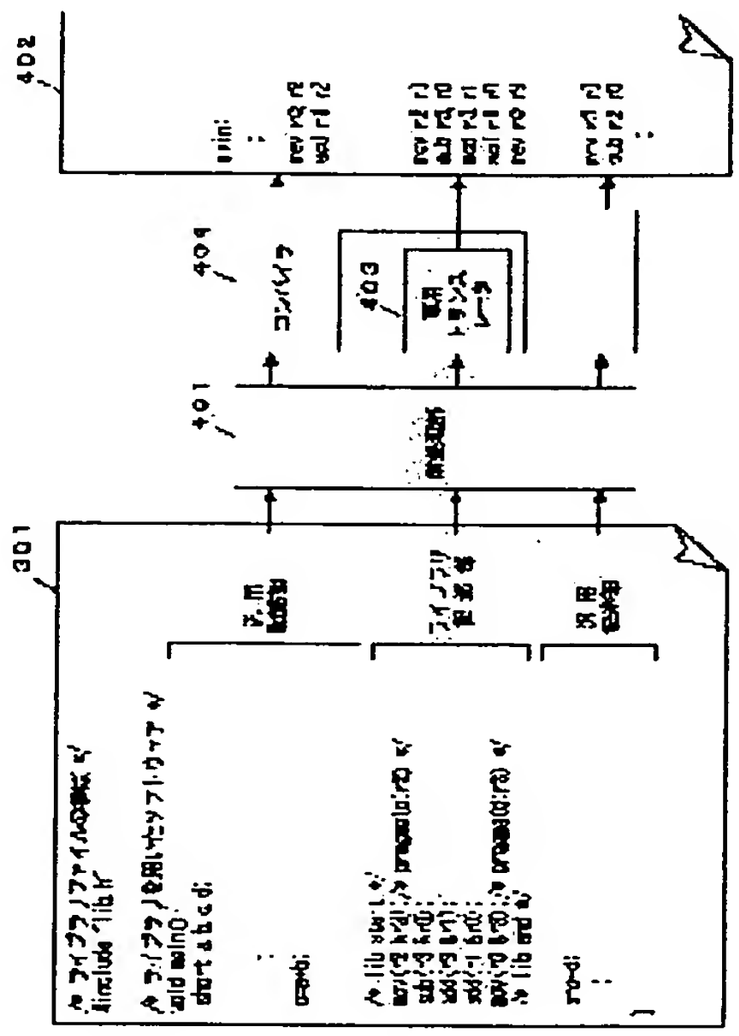
51)Int.Cl. G06F 9/45

21)Application number : 2002-161486
22)Date of filing : 03.06.2002

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD
(72)Inventor : KONDO TAKAHIRO
NAKAMURA TAKESHI
TARUKI MAIKO

## 54) ENVIRONMENT FOR SOFTWARE DEVELOPMENT, SIMULATOR, AND RECORDING MEDIUM

57)Abstract:
'ROBLEM TO BE SOLVED: To provide a technique which eliminates the need for an valuation board, etc., and can perform verification at higher speed than a conventional echnique.
;OLUTION: A source code is described all in a high-level language and includes a library escription part consisting of functions etc. defined in the high-level language and a eneral description part other than the library description part corresponding to an ssembler code of a target processor. Software for the target processor is generated by onverting the library description part into an assembler code by a dedicated translator 03 one to one and compiling the general description part by a compiler 404. A compiler 06 of a simulator compiles the library description part by reference to a library.



:GAL STATUS
)ate of request for examination] 03.06.2005
)ate of sending the examiner's decision of rejection]
Kind of final disposal of application other than the examiner's
ecision of rejection or application converted registration]
)ate of final disposal for application]
)atent number]
)ate of registration]
lumber of appeal against examiner's decision of rejection]
)ate of requesting appeal against examiner's decision of rejection]
)ate of extinction of right]

(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-13190
(P2004-13190A)
(43)公開日　平成16年1月15日(2004.1.15)

| (51) Int.Cl.[7] | F I | | テーマコード（参考） |
|---|---|---|---|
| G06F 9/45 | G06F 9/44 | 322E | 5B081 |

審査請求 未請求　請求項の数 17　OL　（全 19 頁）

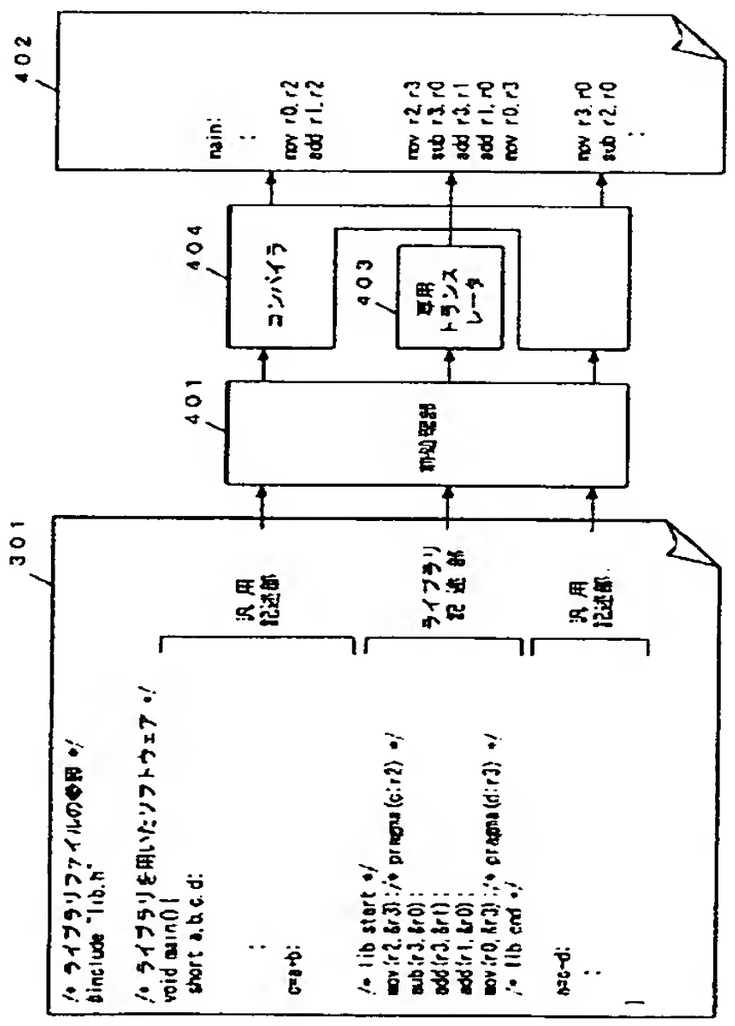
(21) 出願番号　特願2002-161486 (P2002-161486)
(22) 出願日　平成14年6月3日 (2002.6.3)

(71) 出願人　000005821
松下電器産業株式会社
大阪府門真市大字門真１００６番地
(74) 代理人　100097179
弁理士　平野　一幸
(72) 発明者　近藤　孝宏
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72) 発明者　中村　剛
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72) 発明者　樽木　麻衣子
大阪府門真市大字門真１００６番地　松下電器産業株式会社内

Fターム(参考)　5B081　CC41

(54)【発明の名称】ソフトウエア開発環境、シミュレータ及び記録媒体

(57)【要約】
【課題】評価ボード等が必要なく、かつ、従来技術よりも高速な検証が行える技術を提供する。
【解決手段】ソースコードは、全て高級言語で記述され、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数等からなるライブラリ記述部と、ライブラリ記述部以外の汎用記述部とを含む。ターゲットプロセッサ用のソフトウエアを生成するには、ライブラリ記述部を、専用トランスレータ４０３でアセンブラコードに、一対一に変換し、汎用記述部を、コンパイラ４０４によりコンパイルする。シミュレータのコンパイラ９０６は、ライブラリ記述部を、ライブラリを参照してコンパイルする。
【選択図】　図４



【特許請求の範囲】
【請求項１】
高級言語で記述されたソースコードを読み込んで、ターゲットプロセッサ用のアセンブラコードを生成すると共に、このターゲットプロセッサとは異なるホストプロセッサ上で動作するコンパイラを備え、
前記ソースコードは、
ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、
高級言語で記述され、かつ前記ライブラリ記述部に該当しない汎用記述部とを含む、ソフトウエア開発環境。
【請求項２】
前記ライブラリ記述部は、専用トランスレータによりターゲットプロセッサ用のアセンブラコードに、一対一に変換され、かつ、前記汎用記述部は、前記コンパイラによりコンパイルされて、ターゲットプロセッサ用のアセンブラコードが生成される、請求項１記載のソフトウエア開発環境。
【請求項３】
ソースコードは、変数の割付に関する情報を含み、かつ前記コンパイラは、この情報を反映した、ターゲットプロセッサ用のアセンブラコードを生成する、請求項２記載のソフトウエア開発環境。
【請求項４】
ソースコードにおいて、前記ライブラリ記述部と前記汎用記述部とは、前記コンパイラによりコメントと解釈される情報により区別されている、請求項１から３記載のソフトウエア開発環境。
【請求項５】
ソースコードにおいて、前記ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、前記汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、前記コンパイラによりコメントと解釈される書式で記述されている、請求項１から４記載のソフトウエア開発環境。
【請求項６】
変数の割付に関する情報は、前記コンパイラによりコメントと解釈される書式で記述されている、請求項３記載のソフトウエア開発環境。
【請求項７】
ソースコードには、前記ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている請求項１から６記載のソフトウエア開発環境。
【請求項８】
請求項１から７記載のソフトウエア開発環境を実現するプログラムが記録された記録媒体。
【請求項９】
高級言語で記述されたソースコードを読み込んで、このソースコードによる、ターゲットプロセッサの動作をシミュレートし、かつ、ホストプロセッサ上で動作するオブジェクトコードを生成するコンパイラを備え、
それ自体ホストプロセッサ上で動作する、シミュレータであって、
前記ソースコードは、
ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、
高級言語で記述され、かつ前記ライブラリ記述部に該当しない汎用記述部とを含み、
前記ライブラリ記述部で使用される関数又は手続きの定義を含むライブラリが、前記コンパイラにより参照可能に用意されており、
前記コンパイラは、前記汎用記述部をコンパイルすると共に、前記ライブラリ記述部を、

このライブラリを参照してコンパイルして、ホストプロセッサ用のアセンブラコードを生成し、
生成されたアセンブラコードを、ホストプロセッサ上で実行可能なオブジェクトコードに変換する、シミュレータ。
【請求項１０】
前記ライブラリ記述部は、変数の割付に関する情報を含み、この情報に基づいて、前記ライブラリ記述部と前記汎用記述部の接続部を生成し、前記オブジェクトコードに付加する、請求項９記載のシミュレータ。
【請求項１１】
前記ライブラリには、ターゲットプロセッサの割り込み機能が備えられ、ホストプロセッサ上でのシミュレーション時に、この割り込み機能のオン／オフを制御可能とした、請求項９から１０記載のシミュレータ。
【請求項１２】
前記ライブラリには、ターゲットプロセッサの実行サイクル数カウント機能が備えられ、ホストプロセッサ上でのシミュレーション時にターゲットプロセッサのサイクル数計測を可能とした、請求項９から１１記載のシミュレータ。
【請求項１３】
ソースコードにおいて、前記ライブラリ記述部と前記汎用記述部とは、前記コンパイラによりコメントと解釈される情報により区別されている、請求項９から１２記載のシミュレータ。
【請求項１４】
ソースコードにおいて、前記ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、前記汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、前記コンパイラによりコメントと解釈される書式で記述されている、請求項９から１３記載のシミュレータ。
【請求項１５】
変数の割付に関する情報は、前記コンパイラによりコメントと解釈される書式で記述されている、請求項１０記載のシミュレータ。
【請求項１６】
ソースコードには、前記ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている請求項９から１５記載のシミュレータ。
【請求項１７】
請求項９から１６記載のシミュレータを実現するプログラムが記録された記録媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は、ターゲットプロセッサ向け組み込みソフトウエアの開発に適した、ソフトウエア開発環境及びその関連技術に関するものである。
【０００２】
【従来の技術】
ターゲットプロセッサ向け組み込みソフトウエアを開発する際、一般に、ターゲットプロセッサとは互換性のない、ホストプロセッサ上で動作するソフトウエア開発環境が用いられる。
【０００３】
ここで、本明細書において、「ホストプロセッサ」とは、このソフトウエア開発環境を動作させ、開発と、開発の結果物であるソフトウエアの検証とに、使用するプロセッサをいう。
【０００４】
また、「ターゲットプロセッサ」とは、ホストプロセッサとは異なるプロセッサであり、

この開発の結果物であるソフトウエアを実行するプロセッサをいう。

【０００５】

そして、ホストプロセッサとターゲットプロセッサとは、ソフトウエア互換性がない。また、このような、結果物としてのソフトウエアは、専らターゲットプロセッサ上でのみ正常に動作し、ホストプロセッサ上では正常に動作しない。

【０００６】

また、本明細書にいう「シミュレータ」が生成するソフトウエアは、上記結果物であるソフトウエアを模したソフトウエアであるが、ターゲットプロセッサ上で動作するものではなく、ホストプロセッサ上で動作するものである。

【０００７】

このように、ホストプロセッサのソフトウエア開発環境を用いて、ホストプロセッサとは異なるターゲットプロセッサ向けのソフトウエアを開発する場合、次に述べるような問題点がある。

【０００８】

ここで近年、このようなソフトウエアを開発する場合であっても、ソフトウエアが大規模になるに伴い、アセンブラ言語主体による開発から、Ｃ／Ｃ＋＋等の高級言語による開発へ、シフトしている。

【０００９】

これは、高級言語を用いたソフトウエア開発を行えば、データの保持、転送、演算等の処理を、ターゲットプロセッサのアセンブラレベルの命令やレジスタ、メモリ等のリソースに依存しない形式で記述でき、可読性、汎用性、開発効率に優れているためである。

【００１０】

ここで特に、組み込み向けソフトウエア等では、システムのパフォーマンスを最大限引き出すため、プロセッサの能力を極限まで高める、最適化が求められる。

【００１１】

しかしながら、高級言語によるソフトウエア開発では、コンパイラ性能の問題から高級言語からアセンブラコードに変換する際に、冗長なコードが生成され、ソフトウエアのコードサイズや実行速度に影響を及ぼす場合がある。

【００１２】

そこで現在でも、信号処理などの処理負荷が大きい部分に関しては、高級言語による開発に加え、ターゲットプロセッサのアセンブラ言語による開発が行われている。

【００１３】

そのためソフトウエア開発者は、図１１（ａ）のソースコード例１１０１が示すように、多くのコンパイラがサポートしている、高級言語とアセンブラ言語の混在記述仕様"ａｓｍ文"等を用いたり、図１１（ｂ）に示すように、高級言語によるソースコード例１１０２の他に、同図のアセンブラ例１１０３に示すように、プログラムの一部（例えば一つの関数）を、アセンブラのみで記述する開発方法をとっている。

【００１４】

そして、このように記述されたソースコードを、コンパイラを用いてアセンブラコードに翻訳する。このとき、"ａｓｍ文"等やアセンブラで記述された部分についても、コンパイラに解釈され、コンパイラは、アセンブラで記述された部分については、アセンブラコードをそのまま出力する。

【００１５】

そして、アセンブラコードをターゲットプロセッサ上で実行可能なオブジェクトコードに変換する。上述したように、このオブジェクトコードは、ターゲットプロセッサ上でのみ動作し、ホストプロセッサ上では動作しない。

【００１６】

また、ソフトウエアの開発を進めるには、結果物であるソフトウエアが、要求されている仕様を満たすか否か、検証を行う必要がある。ここで、ホストプロセッサ上で、結果物であるソフトウエア（ターゲットプロセッサ用のソフトウエアそのもの）を動作させること

ができれば、話は簡単であるが、上述のように、これはできない。したがって、その他の検証法によらざるを得ない。

【００１７】

従来技術において、この検証を行うには、（検証法１）ターゲットプロセッサ上で実際に実行させる、または、（検証法２）ホストプロセッサ上で動作するシミュレータにより、ターゲットプロセッサの動作をシミュレートする、という手法が、主にとられる。

【００１８】

【発明が解決しようとする課題】

ここで、（検証法１）により、結果物であるソフトウエアを、ターゲットプロセッサ上で実行させようとするには、ターゲットプロセッサ等を物理的に実装した評価ボード等を、ソフトウエア開発者の数だけ用意する必要があり、コスト的に課題がある。また、このような評価ボード等を入手できないときには、この検証法では、全く検証を行えない。

【００１９】

また、（検証法２）により、ホストプロセッサ上で動作するシミュレータを用いると、シミュレーション速度が遅い、という課題がある。

【００２０】

これは、従来のこの種のシミュレータは、インタープリタ型のものしかなく、このシミュレータは、実行するソフトウエアをメモリ内に格納し、実行時に、１命令ずつメモリから読み出し、その命令を解読し、実行するという、プロセスを経るからである。

【００２１】

そこで本発明は、評価ボード等が必要なく、かつ、従来技術よりも高速な検証が行える技術を提供することを目的とする。

【００２２】

より具体的には、本発明は、この目的に合致し、ターゲットプロセッサ用のアセンブラコードを生成する、ソフトウエア開発環境を提供する。また本発明は、この目的に合致し、ホストプロセッサ上で動作するオブジェクトコードを生成する、コンパイラ型のシミュレータを提供する。

【００２３】

【課題を解決するための手段】

第１の発明に係るソフトウエア開発環境は、高級言語で記述されたソースコードを読み込んで、ターゲットプロセッサ用のアセンブラコードを生成すると共に、このターゲットプロセッサとは異なるホストプロセッサ上で動作するコンパイラを備え、ソースコードは、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、高級言語で記述され、かつライブラリ記述部に該当しない汎用記述部とを含む。

【００２４】

この構成により、ソフトウエア開発環境と、シミュレータとに、共通のソースコードを適用できる。また、ソースコードが全て高級言語で記述されるため、移植性や可読性に優れた、開発を行える。

【００２５】

第２の発明に係るソフトウエア開発環境では、ライブラリ記述部は、専用トランスレータによりターゲットプロセッサ用のアセンブラコードに、一対一に変換され、かつ、汎用記述部は、コンパイラによりコンパイルされて、ターゲットプロセッサ用のアセンブラコードが生成される。

【００２６】

この構成により、ライブラリ記述部における、アセンブラレベルでのコード最適化を、生成するアセンブラコードに、そのまま反映できる。

【００２７】

第３の本発明に係るソフトウエア開発環境では、ソースコードは、変数の割付に関する情報を含み、かつコンパイラは、この情報を反映した、ターゲットプロセッサ用のアセンブ

ラコードを生成する。
【００２８】
この構成により、変数の割付に関する指定を、アセンブラコードに反映できる。
【００２９】
第４の発明に係るソフトウエア開発環境では、ソースコードにおいて、ライブラリ記述部と汎用記述部とは、コンパイラによりコメントと解釈される情報により区別されている。
【００３０】
この構成において、ライブラリ記述部は、コンパイラによりコメントと解釈される情報により、汎用記述部と区別されているため、この区別のための情報は、コンパイラにより無視され、コンパイラ型のシミュレータによるシミュレーション時と、ソースコードを共通にしても、不都合を生じない。
【００３１】
第５の発明に係るソフトウエア開発環境では、ソースコードにおいて、ライブラリ記述部は、開始識別子と終了識別子とに挟まれることにより、汎用記述部と区別され、これら開始識別子及び終了識別子は、いずれも、コンパイラによりコメントと解釈される書式で記述されている。
【００３２】
この構成において、ライブラリ記述部は、開始識別子と終了識別子とに挟まれているため、ライブラリ記述部と、汎用記述部とを、明確に区別できる。
【００３３】
第６の発明に係るソフトウエア開発環境では、変数の割付に関する情報は、コンパイラによりコメントと解釈される書式で記述されている。
【００３４】
この構成において、変数の割付に関する情報が、コンパイラによりコメントと解釈される書式で記述されているため、この記述は、コンパイラにより無視され、コンパイラ型のシミュレータによるシミュレーション時と、ソースコードを共通にしても、不都合を生じない。
【００３５】
第７の発明に係るソフトウエア開発環境では、ソースコードには、ライブラリ記述部で使用される関数又は手続きを定義したライブラリをインクルードするコンパイラ疑似命令が含まれている。
【００３６】
この構成により、シミュレーション時のソースコードと、完全なコード共通性を確保できる。なお、ソースコードに、このコンパイラ疑似命令を含めても、コンパイラの処理上、不都合を生じない。
【００３７】
第８の発明に係るシミュレータは、高級言語で記述されたソースコードを読み込んで、このソースコードによる、ターゲットプロセッサの動作をシミュレートし、かつ、ホストプロセッサ上で動作するオブジェクトコードを生成するコンパイラを備え、それ自体ホストプロセッサ上で動作する、シミュレータであり、ソースコードは、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された関数又は手続きを使用して表現されるライブラリ記述部と、高級言語で記述され、かつライブラリ記述部に該当しない汎用記述部とを含み、ライブラリ記述部で使用される関数又は手続きの定義を含むライブラリが、コンパイラにより参照可能に用意されており、コンパイラは、汎用記述部をコンパイルすると共に、ライブラリ記述部を、このライブラリを参照してコンパイルして、ホストプロセッサ用のアセンブラコードを生成し、生成されたアセンブラコードを、ホストプロセッサ上で実行可能なオブジェクトコードに変換する。
【００３８】
この構成により、コンパイラ型のシミュレータを実現できる。しかも、このソースコードは、ソフトウエア開発環境に用いるものと、共通のものを利用できる。コンパイラ型のシ

ミュレータであるから、従来のインタープリタ型のシミュレータに比べ、遙かに高速なオブジェクトコードが得られ、効果的な検証を行える。

【００３９】

第９の発明に係るシミュレータでは、ライブラリ記述部は、変数の割付に関する情報を含み、この情報に基づいて、ライブラリ記述部と汎用記述部の接続部を生成し、オブジェクトコードに付加する。

【００４０】

この構成により、変数の整合性を確保したオブジェクトコードを、生成できる。

【００４１】

第１０の発明に係るシミュレータでは、ライブラリには、ターゲットプロセッサの割り込み機能が備えられ、ホストプロセッサ上でのシミュレーション時に、この割り込み機能のオン／オフを制御可能とした。

【００４２】

この構成により、割り込み機能が必要な場合は当該機能を使用したシミュレーションを行い、割り込み機能が不要なシミュレーションの場合は高速なシミュレーションを行える。

【００４３】

第１１の発明に係るシミュレータでは、ライブラリには、ターゲットプロセッサの実行サイクル数カウント機能が備えられ、ホストプロセッサ上でのシミュレーション時にターゲットプロセッサのサイクル数計測を可能とした。

【００４４】

この構成により、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照できる。

【００４５】

【発明の実施の形態】

以下、図面を参照しながら、本発明の実施の形態を説明する。なお以下、高級言語として、Ｃ言語を取り上げるが、Ｐａｓｃａｌ等他の高級言語を適用することもできる。

（実施の形態１）

以下、本発明の実施の形態１について、図１～図４を参照しながら説明する。本形態は、ソフトウエア開発環境に関する。

【００４６】

図１（ａ）は、本発明の実施の形態１で取り上げるターゲットプロセッサのブロック図である。

【００４７】

図１（ａ）に示すように、本ターゲットプロセッサは、リソースとして４本の１６ビットレジスタｒ０、ｒ１、ｒ２、ｒ３を持ち、演算器として、２つの符号付１６ビットデータを入力し、その加算／減算結果である符号付１６ビットデータを出力する、加減算器１０１を備える。

【００４８】

ここで、加減算器１０１は、演算結果が、符号付１６ビットの最大値（０×７ＦＦＦ）よりも大きい値になった場合は、演算結果を符号付１６ビットの最大値（０×７ＦＦＦ）とし、演算結果が、符号付１６ビットの最小値（０×８０００）よりも小さい値になった場合は、符号付１６ビットの最小値（０×８０００）にクリップする、ＭＡＸＭＩＮ機能を、装備しているものとする。

【００４９】

なお、本形態では、ターゲットプロセッサとして、ＤＳＰを想定しているため、ターゲットプロセッサは、ＤＳＰではあたりまえのＭＡＸＭＩＮ機能を、持つことを前提としているが、本発明が予定するターゲットプロセッサとしては、ＭＡＸＭＩＮ機能を有するものに限定されない。

【００５０】

さて、加減算器１０１と、４本のレジスタｒ０～ｒ３とは、バス１０２、１０３、１０４

により接続され、４本のレジスタｒ０～ｒ３のうちの２本は、加減算器１０１への入力レジスタとして使用され、１本は、加減算器１０１からの結果を格納する出力レジスタとして使用される。なお、入力レジスタ及び出力レジスタとして、どのレジスタを使用するかは、後述するアセンブラ命令により指定される。

【００５１】

図１（ｂ）は、同ターゲットプロセッサ上で実行可能なアセンブラレベルの命令及びその機能を示す説明図である。

【００５２】

図１（ｂ）に示すように、本ターゲットプロセッサは、３つのアセンブラ命令、"ｍｏｖ命令"、"ａｄｄ命令"、"Ｓｕｂ命令"に対応している。勿論、対応する命令を適宜変更しても、本発明に包含される。

【００５３】

ここで、これら各命令の記述形式及び機能について説明する。まず、転送を行う、"ｍｏｖ命令"は"ｍｏｖ　Ｓｒｃ，ｄＳｔ"と記述する。ここで、"Ｓｒｃ"及び"ｄＳｔ"は、４本のレジスタｒ０～ｒ３から選択する。例えば"ｍｏｖ　ｒ０，ｒ１"といった具合である。この命令によって、レジスタｒ０に格納されているデータが、レジスタｒ１に転送される。

【００５４】

次に、加算を行う、"ａｄｄ命令"は"ａｄｄ　Ｓｒｃ，ｄＳｔ"と記述する。ここで、"Ｓｒｃ"及び"ｄＳｔ"は、"ｍｏｖ命令"と同様である。例えば、"ａｄｄ　ｒ０，ｒ１"という命令では、レジスタｒ０、ｒ１に格納されたデータが、加算され、結果がレジスタｒ１に格納される。

【００５５】

また、減算を行う、"Ｓｕｂ命令"は"Ｓｕｂ　Ｓｒｃ，ｄＳｔ"と記述する。ここで"Ｓｒｃ"及び"ｄＳｔ"は、"ｍｏｖ命令"、"ａｄｄ命令"と同様である。例えば、"Ｓｕｂ　ｒ０，ｒ１"という命令では、レジスタｒ０に格納されたデータからレジスタｒ１に格納されたデータが減算され、結果がレジスタｒ１に格納される。

【００５６】

図２は、図１のターゲットプロセッサに対応するライブラリをＣ言語で記述した例示図である。本例では、このライブラリは、"ｌｉｂ．ｈ"というファイル名を持つ１つのファイルからなるが、複数のファイルに分けてライブラリを記述することもできる。

【００５７】

図２に示すように、本形態で使用するライブラリは、ターゲットプロセッサのリソースである４本のレジスタ（ｒ０、ｒ１、ｒ２、ｒ３）の機能を実現する変数（ｒ０、ｒ１、ｒ２、ｒ３）の定義と、ターゲットプロセッサが対応している各アセンブラ命令（ｍｏｖ命令、ａｄｄ命令、Ｓｕｂ命令）の機能を実現する、Ｃ言語の関数（ｍｏｖ関数、ａｄｄ関数、Ｓｕｂ関数）の定義とを、含む。

【００５８】

まず、ターゲットプロセッサのレジスタ（ｒ０～ｒ３）は、１６ビットであるため、ライブラリにおいて、これらのレジスタに対応する、変数（ｒ０～ｒ３）は、１６ビットデータ型"Ｓｈｏｒｔ"で宣言する。また、ライブラリにおいて、それ以外の演算精度等も、ターゲットプロセッサと合致するように記述する。

【００５９】

またここで、各アセンブラ命令（ｍｏｖ命令、ａｄｄ命令、Ｓｕｂ命令）と、ライブラリにおいて、これらの命令に対応するＣ言語の関数（ｍｏｖ関数、ａｄｄ関数、Ｓｕｂ関数）とは、一対一に対応している。なお、引数の内容及び順序も、一致させておくことが望ましい。

【００６０】

そして、これらＣ言語の関数は、引数に演算に使用するレジスタ（変数ｒ０～ｒ３）を指定して呼び出され、指定されたレジスタに格納されたデータに対して各種演算を行い、結

果を変数にセットして戻る構成となっている。
【００６１】
ターゲットプロセッサにおいて、レジスタ間のデータ転送命令であるｍｏｖ命令機能を実現するｍｏｖ関数は、第１の引数で渡された転送元のレジスタ値（変数ｓｒｃ値）を第２の引数で渡されたレジスタ（変数＊ｄｓｔ）へ転送する。
【００６２】
また、加算命令であるａｄｄ命令機能を実現するａｄｄ関数は、第１及び第２の引数で渡された入力レジスタ値（変数ｓｒｃ、＊ｄｓｔ値）に対し加算処理並びにＭＡＸＭＩＮ処理を実行し、結果を第２の引数で渡されたレジスタ（変数＊ｄｓｔ）に格納する。
【００６３】
減算命令であるｓｕｂ命令機能を実現するｓｕｂ関数も、同様である。
【００６４】
なお、Ｃ言語の仕様上、戻り値のない（ｖｏｉｄ型）の処理も、”手続き”とは呼ばず、”関数”と呼ぶ習慣があるため、図２のライブラリは、変数と関数の定義からなる。
【００６５】
しかしながら、他の高級言語、例えばＰａｓｃａｌでは、戻り値のない手続き（Ｐｒｏｃｅｄｕｒｅ）と、戻り値を持つ関数（ｆｕｎｃｔｉｏｎ）とを、厳格に区別している。また、本明細書にいう高級言語には、Ｃ言語以外の高級言語、例えばＰａｓｃａｌ等も、含めている。したがって、本明細書にいうライブラリでは、アセンブラコードに対応する処理は、一般に、”関数又は手続きにより高級言語で定義されている”ということになる。
【００６６】
図３は、本発明の実施の形態１におけるソースコードの例示図である。以下、ソースコードにおいて、ターゲットプロセッサのアセンブラコードに対応して、高級言語で定義された、関数又は手続きで表現される部分を「ライブラリ記述部」といい、ライブラリ記述部以外の部分を「汎用記述部」という。
【００６７】
図３に示すように、このソースコード３０１では、まず先頭で、ライブラリファイル（”ｌｉｂ．ｈ”）が、インクルードされる。
【００６８】
また、ライブラリ記述部では、このライブラリにおいて定義された、変数や関数を用いて、プログラムが記述される。
【００６９】
図３において、”／＊　ｌｉｂ＿ｓｔａｒｔ　＊／”は、ライブラリ記述部の開始識別子であり、”／＊　ｌｉｂ＿ｅｎｄ　＊／”は、ライブラリ記述部の終了識別子である。
【００７０】
このように、本形態では、ライブラリ記述部は、開始識別子と終了識別子とで挟まれることにより、汎用記述部と、明確に区別される。
【００７１】
しかも、これら開始識別子及び終了識別子は、いずれもＣ言語のコンパイラにより、コメントと解釈される書式となっている。さらに、ライブラリ記述部には、”ａｓｍ文”等の標記は、行われていない。
【００７２】
よって、このソースコード３０１を、Ｃ言語のコンパイラに通すと、汎用記述部だけでなく、ライブラリ記述部についても、コンパイラは、Ｃ言語そのもの（ａｓｍ文等は除く）で記述されていると解釈する。
【００７３】
より具体的には、コンパイラが、”ｍｏｖ（ｒ２．＆ｒ３）；”という行を処理するとき、これをＣ言語の関数として処理し、文頭でインクルードされている、ライブラリファイル”ｌｉｂ．ｈ”にある、ｍｏｖ関数の定義を、この行に、あてはめる。
【００７４】

同様に、ソースコード３０１の全部を、そのままコンパイラに通せば、コンパイラは、ｍｏｖ、Ｓｕｂ、ａｄｄ等についても、そういう識別子を持つＣ言語の関数として解釈するから、ソースコード３０１の全部を、何ら問題なく、コンパイル処理できる。

【００７５】

しかしながら、このような取り扱いは、後述する実施の形態２における、コンパイラ型のシミュレータに係るものであり、実施の形態１における、ソフトウエア開発環境では、ライブラリ記述部を、コンパイラに通さないように、後に詳述する前処理を行う。

【００７６】

また、”／＊　Ｐｒａｇｍａ（ｃ：ｒ２）　＊／”や”／＊　Ｐｒａｇｍａ（ｄ：ｒ３）　＊／”は、後述するコンパイラにおいて、変数ｃを、ターゲットプロセッサのレジスタｒ２に割り付け、変数ｄを、レジスタｒ３に割り付けるように、指定するキーワードである。

【００７７】

図４は、本発明の実施の形態１におけるソフトウエア開発環境の機能ブロック図である。

【００７８】

図４において、前処理部４０１は、コンパイラ４０４等の処理に先立ち、ソースコード３０１を、”／＊　ｌｉｂ＿ｓｔａｒｔ　＊／”という開始識別子と、”／＊　ｌｉｂ＿ｅｎｄ　＊／”という終了識別子とに挟まれた、ライブラリ記述部と、それ以外の汎用記述部とに、分離する。

【００７９】

そして、前処理部４０１は、ソースコード３０１の文頭から順に、汎用記述部については、コンパイラ４０４に渡して、アセンブラコードを生成させ、ライブラリ記述部については、専用トランスレータ４０３へ渡し、コンパイラ４０４を経由せずに、アセンブラコードを生成させる。これにより、アセンブラコードリスト４０２が、生成される。

【００８０】

また、専用トランスレータ４０３は、”ｍｏｖ（ｒ２，＆ｒ３）；”等のソースコードを、ターゲットプロセッサ用のアセンブラコードに、一対一に変換（”ｍｏｖ（ｒ２，ｒ３）関数”１”ｍｏｖ　ｒ２，ｒ３命令”、”Ｓｕｂ（ｒ３，ｒ０）関数”１”Ｓｕｂ　ｒ３，ｒ０命令”等）する。

【００８１】

ここで、”／＊　Ｐｒａｇｍａ（ｃ：ｒ２）　＊／”や”／＊　Ｐｒａｇｍａ（ｄ：ｒ３）　＊／”等による、レジスタ割付の指示も、アセンブラコードに反映される。

【００８２】

専用トランスレータ４０３の機能は、テキストベースの単純な置換テーブルを用いれば、容易に実現できる。

【００８３】

また、ソースコード３０１の文頭で、”ｌｉｂ．ｈ”をインクルードする、コンパイラ疑似命令が記述されているが、ライブラリ記述部をコンパイラ４０４に渡さないようにしても、このライブラリにおける関数の定義のあてはめが、なされないだけであって、コンパイラ４０４の処理上、何ら問題ない。

【００８４】

また、このようにすることによって、後述するシミュレータにおいて、ソースコード３０１そのものを、何ら変更を加えることなく、処理できる。

【００８５】

なお、ライブラリ記述部の切り分けキーワードや変数のレジスタ割り付け指定形式は、図示した例に限定されない。例えば、”／＊　ｌｉｂ＿Ｘ＿Ｙ　＊／”等により、Ｘ行目からＹ行目までが、ライブラリ記述部である旨、指示することもできる。また、コンパイラが、対応できるのであれば、ライブラリ記述部の切り分けキーワードや変数のレジスタ割り付け指定形式を、コメントの形式でない形式で記述しても良い。

【００８６】

またコンパイラ４０４による、Ｃ言語からアセンブラコードへの変換方法や、変数のレジスタ割り付け以外の詳細については、本発明の主旨とは関係無いことから説明を省略する。なお、本形態は、ソースコードに汎用記述部が含まれず、ソースコードの全てが、ライブラリ記述部から構成される場合にも、対応できる。

【００８７】

以上のように、本形態のソフトウエア開発環境によれば、ライブラリを用いて開発したソフトウエアを翻訳しアセンブラコードを生成する際に、ライブラリ記述部は、専用トランスレータ４０３で、対応するアセンブラ命令に一対一変換し、汎用記述部は、コンパイラ４０４により、Ｃ言語で記述された処理を実現するアセンブラコードに変換することにより、ライブラリを用いたアセンブラレベルでのコード最適化を、生成するアセンブラコードに、そのまま反映できる。

【００８８】

勿論、図４に示した、ソフトウエア開発環境に、アセンブラコードをオブジェクトコードに変換する機能を追加しても、差し支えない。

【００８９】

（実施の形態２）

本形態は、ターゲットプロセッサ用のソースコード（実施の形態１のソースコード３０１と同じ）を読み込んで、ホストプロセッサ上で動作するオブジェクトコードを生成する、シミュレータに係る。なお、実施の形態１における説明と、重複する部分については、繰り返しを避けるため、説明を省略する。

【００９０】

図５は、本発明の実施の形態２におけるシミュレータの機能ブロック図である。図５において、コンパイラ９０６は、ホストプロセッサ向けのＣ言語コンパイラであり、オブジェクトコード変換部５０３は、コンパイラ９０６が生成したアセンブラコード５０２を、オブジェクトコード５０４へ変換する。

【００９１】

これらコンパイラ９０６、オブジェクトコード変換部５０３は、特殊なものである必要はなく、単に、ホストプロセッサ向けのものであればよい。

【００９２】

但し、ソースコード３０１と、ライブラリ２０１（"ｌｉｂ．ｈ"）は、実施の形態１の説明で述べた内容になっていなければならない。

【００９３】

なおこのとき、ソースコード３０１において、"／＊　ｌｉｂ＿ｓｔａｒｔ　＊／"や"／＊　Ｐｒａｇｍａ（ｃｉｒ２）　＊／"等の表記を省略することもできるが、シミュレータと、ソフトウエア開発環境とにおいて、同一のソースコードを用いることが望ましいので、省略しない方がよい。

【００９４】

また、ここでは、ソースコード３０１の文頭の、"ｌｉｂ．ｈ"をインクルードするコンパイラ疑似命令は、省略できない。なぜなら、これを外すと、コンパイラが、ライブラリ２０１を参照できなくなるからである。

【００９５】

このように、本形態によれば、コンパイラ型のシミュレータを実現でき、ホストプロセッサだけで、ターゲットプロセッサ用のソフトウエアの、シミュレーションができる。

【００９６】

しかも、このシミュレータが生成し、ホストプロセッサ上で動作するソフトウエアは、インタープリタ型のそれよりも、遙かに高速であり、実際のターゲットプロセッサ用のソフトウエアの実行速度に近い、実行速度が得られるため、より効果的な検証を行える。

【００９７】

また、ターゲットプロセッサのアセンブラレベルでの命令コード機能を、Ｃ言語等の高級言語で実現するライブラリを用意することで、アセンブラレベルでのコード最適化（コー

ドサイズ削減や実行時間の高速化等）を含めたソフトウエアを、全てＣ言語等の高級言語で開発できる。

【００９８】

これは、従来アセンブラ言語を用いて開発されたソフトウエアを他のプロセッサへ移植する場合、アセンブラ言語で記述された部分を移植先のプロセッサ向けに修正する必要があったのに対し、Ｃコンパイラに対応したプロセッサであれば、そのまま何の修正も無しに移植が可能であり、ソフトウエアの汎用性を高めるものである。

【００９９】

（実施の形態３）

以下、実施の形態３について、図５～図１０を参照しながら説明する。実施の形態３では、割り込み及び実行サイクル数への対応を、追加する。

【０１００】

図６は、本発明の実施の形態３におけるシミュレータの機能ブロック図である。

【０１０１】

このシミュレータを概説する。ライブラリ５０１は、ターゲットプロセッサのアセンブラレベルの命令コードの機能をＣ言語で定義した、関数群を含む。

【０１０２】

また、ターゲットプロセッサ向けソースコード３０１は、このライブラリ５０１において定義された関数等を用いて記述したライブラリ記述部と、それ以外の汎用記述部とを含む。

【０１０３】

また、アセンブラコード変換部５０７は、Ｃ言語のコンパイラを備え、このコンパイラで、ライブラリ記述部及び汎用記述部を、コンパイルし、ホストプロセッサ上で動作するアセンブラコード５０３を生成する。

【０１０４】

さらに、オブジェクトコード変換部５０９は、生成されたアセンブラコード５０３を、オブジェクトコードに変換し、ホストプロセッサ上で実行可能なオブジェクトコード（ａ．ｏｕｔ＊）５０４が生成され、このコード５０４が、ホストプロセッサ上で実行される。

【０１０５】

以下、図７を用いて、本形態で取り上げるターゲットプロセッサの構成を、具体的に説明する。図７は、本発明の実施の形態３で取り上げるターゲットプロセッサのブロック図である。

【０１０６】

このターゲットプロセッサは、基本的に、実施の形態１に係る図１（ａ）に示した構成と同様の構成を持つが、１６ビットレジスタｉｒが追加されている点が異なる。

【０１０７】

このレジスタｉｒの値は、通常「０」であるが、加減算器６０１で演算結果が符号付１６ビットデータの最大値（０×７ＦＦＦ）よりも大きい値になった場合又は符号付１６ビットデータの最小値（０×８０００）よりも小さい値になった場合に「１」にセットされる。

【０１０８】

そして、レジスタｉｒが「１」にセットされると、実行中のプログラムに割り込みが発生して、レジスタｉｒは「０」にリセットされる。

【０１０９】

割り込み発生後のシーケンスとしては、割り込み処理関数（本形態では、関数名をｉｒＳ（）とする）に制御を移す。

【０１１０】

そして、割り込み処理関数の処理が終了すると、元のプログラムに復帰し処理を続行する。

【０１１１】

ここで、割り込み発生時のターゲットプロセッサのハードウエア的なシーケンスの詳細については、本発明の主旨とは関係無いことから説明を省略する。

【０１１２】

図８は、図６のライブラリ全文をＣ言語で記述した例示図である。

【０１１３】

このライブラリは、基本的に、実施の形態１のもの（図２）と同様であるが、ターゲットプロセッサが備える割り込み機能並びにソフトウエアの実行サイクル数カウント機能が追加されている点が異なる。また、このライブラリファイルの名前は、"ｌｉｂ．ｈ"である。

【０１１４】

このライブラリは、割り込み機能に関し、１６ビットレジスタｉｒ機能を実現する変数ｉｒの定義（"Ｓｈｏｒｔ　ｉｒ＝０；"の部分）と、各アセンブラ命令（ｍｏｖ命令、ａｄｄ命令、Ｓｕｂ命令）の機能を実現する関数群（ｍｏｖ関数、ａｄｄ関数、Ｓｕｂ関数）とを備える。

【０１１５】

また、これらの関数は、その先頭に、変数ｉｒの値が０ではなかったら変数ｉｒの値を０にリセットし、割り込み処理関数（ｉｒＳ（））を呼ぶ機能（"ｉｆ（ｉｒ）｛　｝"の部分）を備える。

【０１１６】

また、ａｄｄ関数及びＳｕｂ関数は、演算結果が符号付１６ビットの最大値（０×７ＦＦＦ）より大きい場合又は符号付１６ビットの最小値（０×８０００）より小さい場合に、変数ｉｒに１をセットする機能（"ｉｒ＝１；"の部分）を備える。

【０１１７】

ここで、ライブラリ内における上記割り込み機能に関する処理を実行するかどうかは、当該処理を条件付コンパイラ疑似命令"＃ｉｆｄｅｆ　＿ＩＲ"及び"＃ｅｎｄｉｆ"で指定しており、後述するコンパイル時に選択可能である。

【０１１８】

また、このライブラリは、ソフトウエアの実行サイクル数カウント機能に関し、サイクル数カウント用の変数ｃｏｕｎｔの定義（初期値は０）（"ｉｎｔ　ｃｏｕｎｔ＝０；"の部分）を備える。また、各関数（ｍｏｖ関数、ａｄｄ関数、Ｓｕｂ関数）は、変数ｃｏｕｎｔをインクリメントする機能（"ｃｏｕｎｔ＋＝１；"の部分、ここで変数ｃｏｕｎｔをインクリメントしている値（１）は各関数が機能を実現するアセンブラ命令（ｍｏｖ命令、ａｄｄ命令、Ｓｕｂ命令）をターゲットプロセッサ上で実行した場合の処理サイクル数である）を備える。

【０１１９】

図９は、図６のアセンブラコード変換部の内部を示す詳細図である。図９に示すように、アセンブラコード変換部５０７は、まずデータ接続用コード生成部９０５を用いて、接続コード挿入済みソースコード９０８を生成する。

【０１２０】

本形態における、接続コード挿入済みソースコード９０８は、ソースコード３０１に対し、ソースコード３０１のライブラリ記述部と汎用記述部の間に、接続コード（ｒ２＝ｃ；ｒ３＝ｄ；等）を、挿入したものである。

【０１２１】

ここで、ソースコード３０１において、ライブラリ記述部では、ライブラリ５０１で定義されている変数（ｒ０～ｒ３）が使用され、汎用記述部では、通常の変数（ａ～ｄ）が使用されている。

【０１２２】

実施の形態１において、このコード９０８を、ターゲットプロセッサ向けに翻訳する際、変数（ｒ０～ｒ３）は、専用トランスレータにより、対応するレジスタ（ｒ０～ｒ３）に割り付けられ、また変数（ａ～ｄ）もコンパイラによりレジスタ（ｒ０～ｒ３）のいずれ

かに自動的に割り付けられる。

【０１２３】

このとき、実施の形態１において、ライブラリ記述部で使用しているレジスタと汎用記述部で使用している変数（ａ～ｄ）の対応付け（どのレジスタにどの変数の値が格納されるか）を意識し、変数（ａ～ｄ）を、どのレジスタに割り付けるかを指定しておけば、アセンブラコードにおけるレジスタ間でのデータの整合性がとられ、意図した処理が実行できる。

【０１２４】

しかしながら、実施の形態２では、ソースコード３０１ではなく、接続コード挿入済みソースコード９０８を、コンパイラを用いて翻訳する。

【０１２５】

以下、データ接続用コード生成部９０５により、ソースコード３０１に接続コードを挿入する点について、説明する。ここで、実施の形態３におけるシミュレータで、ソースコード３０１を、そのままコンパイラ９０６でコンパイルすると、ライブラリ記述部で使用している変数（ｒ０～ｒ３）と、汎用記述部で使用している変数（ａ～ｄ）とが、全く別の変数として扱われてしまい、データの整合性がとれないことになる。

【０１２６】

そこで、実施の形態３では、データ接続用コード生成部９０５において、ライブラリ記述部における変数（ｒ０～ｒ３）と、汎用記述部における変数（ａ～ｄ）との対応付け情報を基に、ソースコード３０１にライブラリ記述部と汎用記述部間の接続コード（ｒ２＝ｃ；ｒ３＝ｄ；等）を挿入し、接続コード挿入済みソースコード９０８を生成し、このソースコード９０８を、コンパイラ９０６でコンパイルすることとしている。

【０１２７】

以下、データ接続用コード生成部９０５の動作を具体的に説明する。図１０は、本発明の実施の形態３におけるデータ接続用コード生成部のフローチャートである。

【０１２８】

まず、データ接続用コード生成部９０５は、ステップ１００１において、ライブラリを用いたソースコード３０１からライブラリ記述部における変数（ｒ０～ｒ３）と汎用記述部における変数（ａ～ｄ）の対応付け情報を取得する。

【０１２９】

図６に示すソースコード３０１では、この変数対応付け情報は、”／＊　Ｐｒａｇｍａ（ｃ：ｒ２）　＊／”、”／＊　Ｐｒａｇｍａ（ｄ：ｒ３）　＊／”として与えられている。

【０１３０】

ステップ１００１では、ソースコード内を”Ｐｒａｇｍａ”というキーワードより検索し、このキーワードの後の”（）”内の”：”で区切られたパラメータを、変数対応付け情報とする。

【０１３１】

ここでは、変数ｃと変数ｒ２、変数ｄと変数ｒ３が対応付けられているものとする。なお、変数対応付け情報の与え方は、図示した例に限定されない。

【０１３２】

次に、ステップ１００２において、データ接続用コード生成部９０５は、ライブラリを用いたソースコード３０１におけるライブラリ記述部の場所を解析する。

【０１３３】

図６に示すソースコード３０１に対し、データ接続用コード生成部９０５は、”／＊　ｌｉｂ＿ｓｔａｒｔ　＊／”という開始識別子の位置を、ライブラリ記述部のスタートポイントとして認識し、”／＊　ｌｉｂ＿ｅｎｄ　＊／”という終了識別子の位置を、ライブラリ記述部のエンドポイントと認識する。

【０１３４】

即ち、ステップ１００２では、データ接続用コード生成部９０５は、ソースコード３０１

内を"ｌｉｂ＿ｓｔａｒｔ"、"ｌｉｂ＿ｅｎｄ"というキーワードで検索し、ライブラリ記述部の場所（スタートポイントとエンドポイント）を解析する。

【０１３５】

次に、ステップ１００３において、データ接続用コード生成部９０５は、ステップ１００２で特定した、ライブラリ記述部のスタートポイントに、ステップ１００１で取得した変数対応付け情報に基づいて、接続コードを挿入する。

【０１３６】

挿入される接続コードは、変数対応付け情報で指定されたライブラリ記述部における変数（ｒ０～ｒ３）に、汎用記述部における変数（ａ～ｄ）のデータを転送するコードである。図９の例では、変数ｃと変数ｒ２、変数ｄと変数ｒ３が対応付けられており、"ｒ２＝ｃ；"、"ｒ３＝ｄ；"が挿入される。

【０１３７】

次に、ステップ１００４において、データ接続用コード生成部９０５は、ステップ１００２で特定した、ライブラリ記述部のエンドポイントに、ステップ１００１で取得した変数対応付け情報に基づいて、接続コードを挿入する。

【０１３８】

挿入される接続コードは、変数対応付け情報で指定された汎用記述部における変数（ａ～ｄ）に、ライブラリ記述部における変数（ｒ０～ｒ３）のデータを転送するコードである。図９の例では、"ｃ＝ｒ２；"、"ｄ＝ｒ３；"が挿入される。

【０１３９】

以上のようにして、データ接続用コード生成部９０５は、ソースコード３０１に対し、上記接続コードを挿入し、接続コード挿入済みソースコード９０８を生成する。

【０１４０】

次に、コンパイラ９０６は、ライブラリ５０１及びソースコード９０８を翻訳し、ホストプロセッサのアセンブラコード９０３を生成し、これを、オブジェクトコード変換部５０９が、実行可能なオブジェクトコード９０４に変換する。

【０１４１】

本発明が予定するコンパイラは、特殊なものである必要はなく、ホストプロセッサに対応した汎用のコンパイラで十分である。例えば、パソコンやワークステーション向けのＦｒｅｅ　Ｓｏｆｔｗａｒｅ　ＦｏｕｎｄａｔｉｏｎのＣコンパイラＧＣＣ等を用いることができる。

【０１４２】

ここで、Ｃコンパイラ（ＧＣＣ）を用いてソースコードを翻訳し、アセンブラコードを生成する際に、オプションとして"－Ｄ＿ＩＲ"を指定すれば、上記割り込み機能を組み込むことができ、また指定しなければ当該機能を外すことが可能である。

【０１４３】

最後に、図６に示すように、生成されたオブジェクトコード（ａ．ｏｕｔ＊）５０４（図５）を、ホストプロセッサ（パソコンやワークステーション等）上で実行することで、ライブラリ内で記述されたターゲットプロセッサの機能（演算、割り込み等）や実行サイクル数カウント機能もホストプロセッサ上で実現でき、ターゲットプロセッサ向けに開発したソフトウエア（アセンブラレベルも含む）のシミュレーションが可能となる。

【０１４４】

また、このシミュレータは、コンパイラ型であり、実行前にコンパイルするため、実行時に実行命令の解読処理などが必要なく、従来のインタプリタ型シミュレータ上でのシミュレータに対して、高速なシミュレーションが可能となる。

【０１４５】

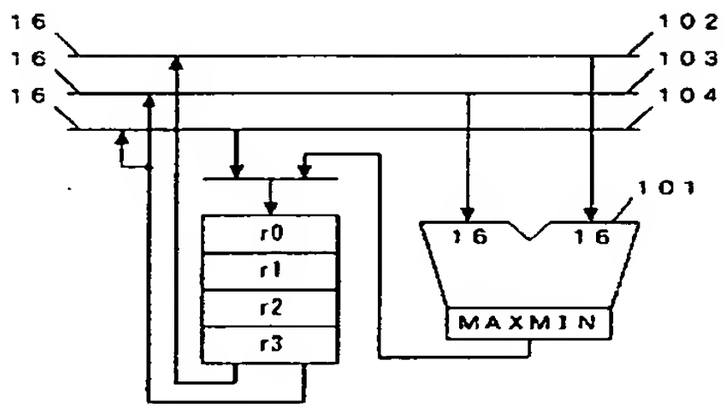
以上のように、本形態によれば、アセンブラレベルでのコード最適化（コードサイズ削減や実行時間の高速化等）を含めたソフトウエアを全てＣ言語等の高級言語で開発し、開発したソフトウエアをホストプロセッサ上でコンパイル、実行でき、従来のシミュレータによるシミュレーションよりも、高速なシミュレーションが可能となる。

【０１４６】
また、ライブラリ記述部で使用している変数（ｒ０～ｒ３）と、それ以外の部分で使用している変数（ａ～ｄ）の対応付けに対応することにより、ライブラリ記述部と汎用記述部において、データの整合性を担保できる。
【０１４７】
また、ライブラリ内にターゲットプロセッサの割り込み機能を備え、かつシミュレーション実行時に、この機能を使用するかどうか選択可能とすることで、割り込み機能が必要な場合、当該機能を使用したシミュレーションを行い、逆に、割り込み機能が不要な場合、当該機能を使用せず高速なシミュレーションを行える。
【０１４８】
また、ライブラリ内で用意したサイクル数カウント変数ｃｏｕｎｔを、例えばシミュレーション終了時にＣ言語における標準出力関数Ｐｒｉｎｔｆ等で表示させたりすることで、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照可能となる。
【０１４９】
【発明の効果】
本発明のソフトウエア開発環境によれば、アセンブラレベルでのコード最適化を、生成するアセンブラコードにそのまま反映できる。
【０１５０】
また、本発明のシミュレータによれば、従来のシミュレータによるシミュレーションよりも、高速なシミュレーションを行える。
【０１５１】
また、ライブラリ記述部と汎用記述部の、変数の対応付けに対応して、これらの間における、データの整合性を担保できる。
【０１５２】
また、ターゲットプロセッサの割り込み機能を備え、割り込み機能のオン／オフに対応して、割り込み機能が必要な場合は当該機能を使用したシミュレーションを行い、割り込み機能が不要なシミュレーションの場合は高速なシミュレーションを行える。
【０１５３】
また、シミュレーション実行時のターゲットプロセッサのアセンブラ命令の実行サイクル数を参照できる。
【図面の簡単な説明】
【図１】（ａ）本発明の実施の形態１で取り上げるターゲットプロセッサのブロック図
（ｂ）同ターゲットプロセッサ上で実行可能なアセンブラレベルの命令及びその機能を示す説明図
【図２】図１のターゲットプロセッサに対応するライブラリをＣ言語で記述した例示図
【図３】本発明の実施の形態１におけるソースコードの例示図
【図４】本発明の実施の形態１におけるソフトウエア開発環境の機能ブロック図
【図５】本発明の実施の形態２におけるシミュレータの機能ブロック図
【図６】本発明の実施の形態３におけるシミュレータの機能ブロック図
【図７】本発明の実施の形態３で取り上げるターゲットプロセッサのブロック図
【図８】図６のライブラリ全文をＣ言語で記述した例示図
【図９】図６のアセンブラコード変換部の内部を示す詳細図
【図１０】同データ接続用コード生成部のフローチャート
【図１１】（ａ）従来のソースコードの例示図
（ｂ）従来のソースコード及びアセンブラコードの例示図
【符号の説明】
２０１、５０１　ライブラリ
３０１　ソースコード
４０１　前処理部

４０３　専用トランスレータ
４０４、９０６　コンパイラ
５０３、５０９　オブジェクトコード変換部
５０７　アセンブラコード変換部

【図１】

(a)




(b)

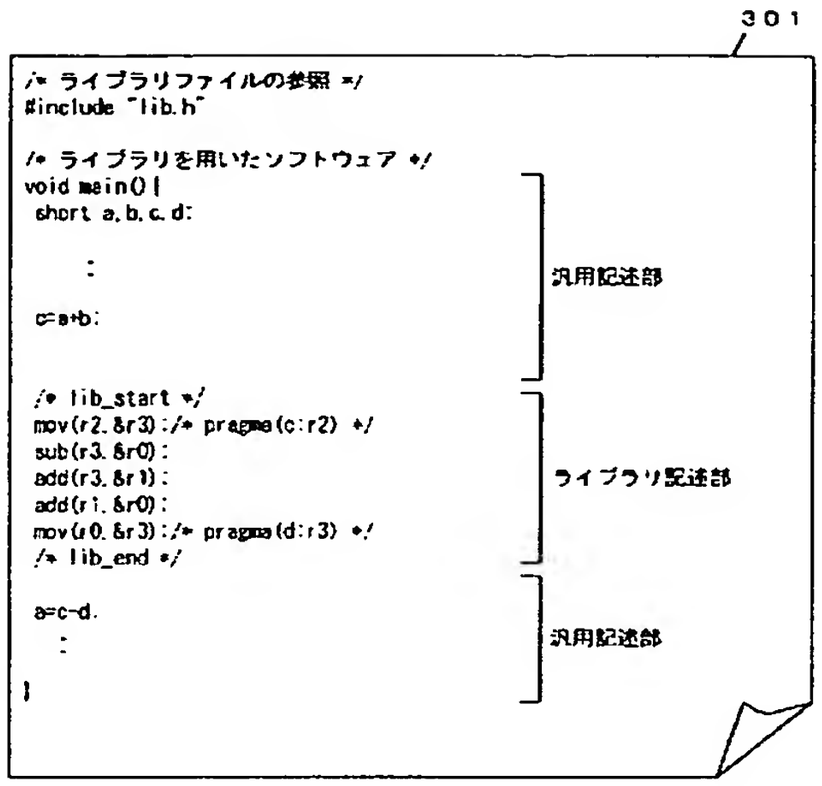
| 命令 | mov | add | sub |
|---|---|---|---|
| 記述形式 | mov src, dst | add src, dst | sub src, dst |
| 機能 | dst←src | dst←src+dst | dst←src-dst |
| 備考 | srcは[r0, r1, r2, r3]中から選択<br>dstは[r0, r1, r2, r3]中から選択 | | |

【図２】

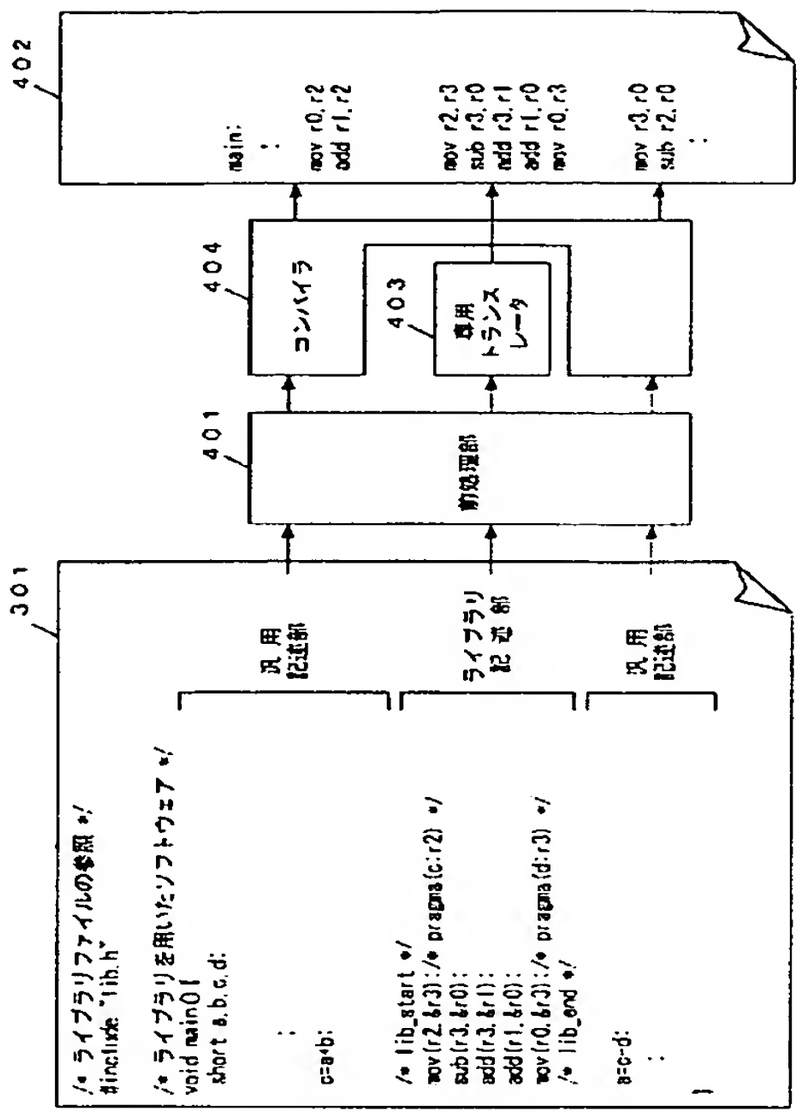
lib.h　201

```
/* レジスタ機能を実現する変数の定義 */
short r0,r1,r2,r3;

/* アセンブラ命令の機能を実現する関数の定義 */
/* mov命令 */
void mov(short src,short *dst){
 /* 転送処理 */
 *dst=src;
}
/* add命令 */
void add(short src,short *dst){
 int tmp;

 /* 加算処理 */
 tmp=(int)src+(int)*dst;

 /* MAXMIN処理 */
 if(tmp>(int)0x00007FFF){
   *dst=(short)0x7FFF;
 }
 else if(tmp<(int)0xFFFF8000){
   *dst=(short)0x8000;
 }
 else{
   *dst=(short)tmp;
 }
}
/* sub命令 */
void sub(short src,short *dst){
 int tmp;

 /* 減算処理 */
 tmp=(int)src-(int)*dst;

 /* MAXMIN処理 */
 if(tmp>(int)0x00007FFF){
   *dst=(short)0x7FFF;
 }
 else if(tmp<(int)0xFFFF8000){
   *dst=(short)0x8000;
 }
 else{
  *dst=(short)tmp;
 }
}
```

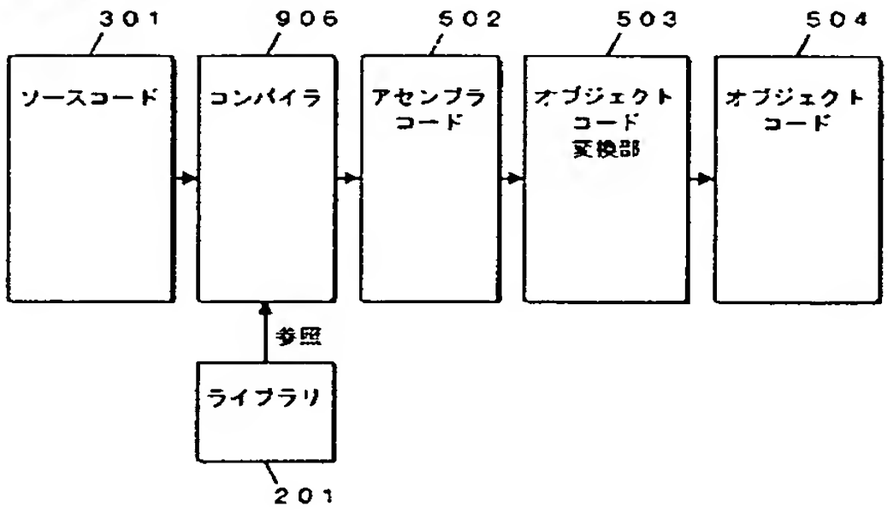
【図３】



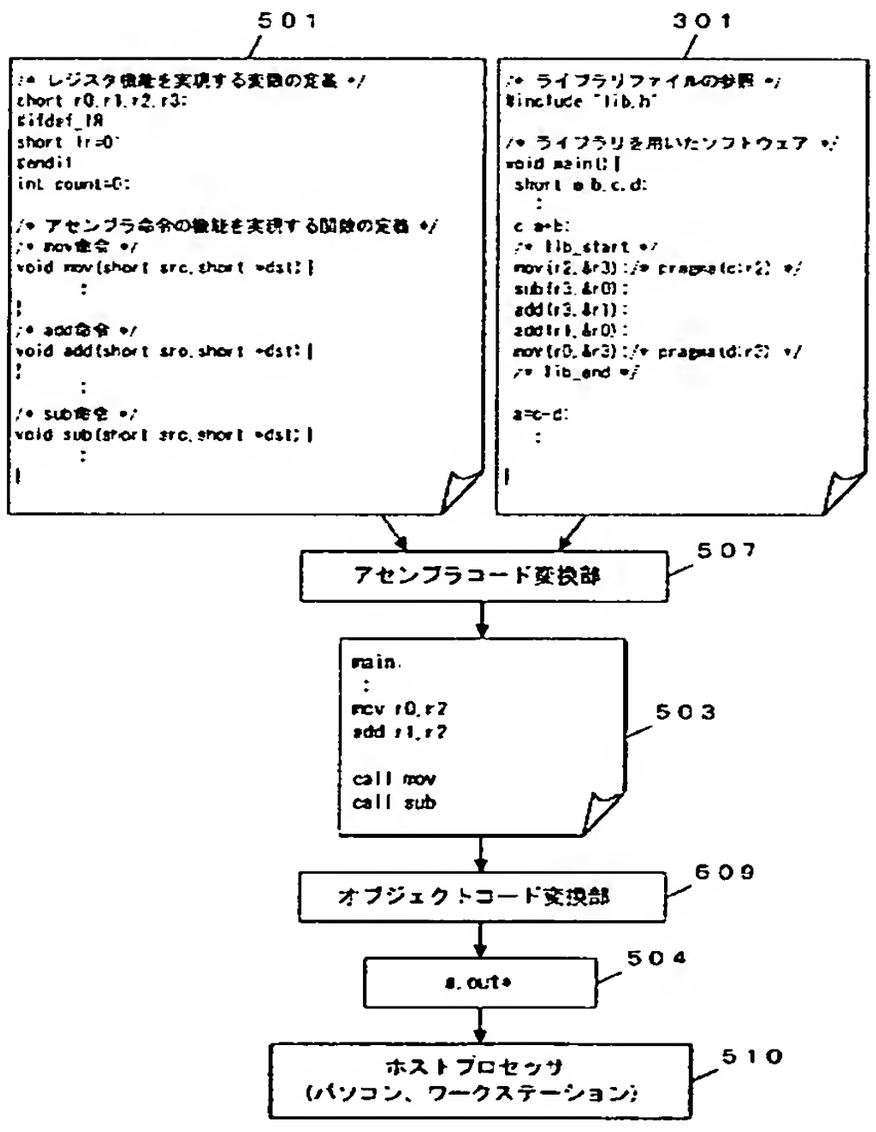

【図４】



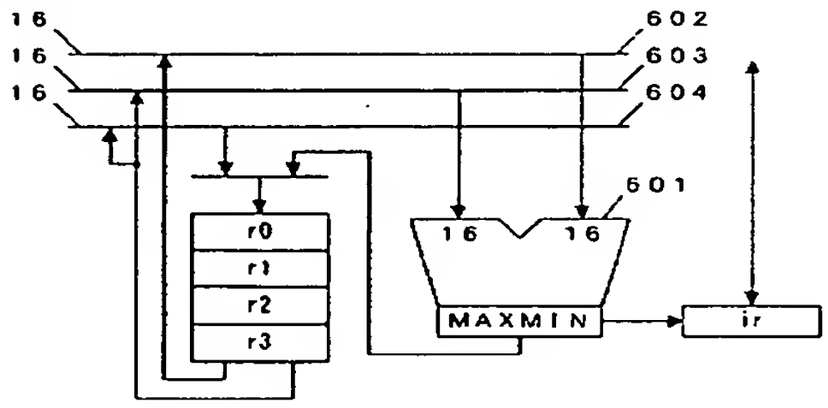

【図５】



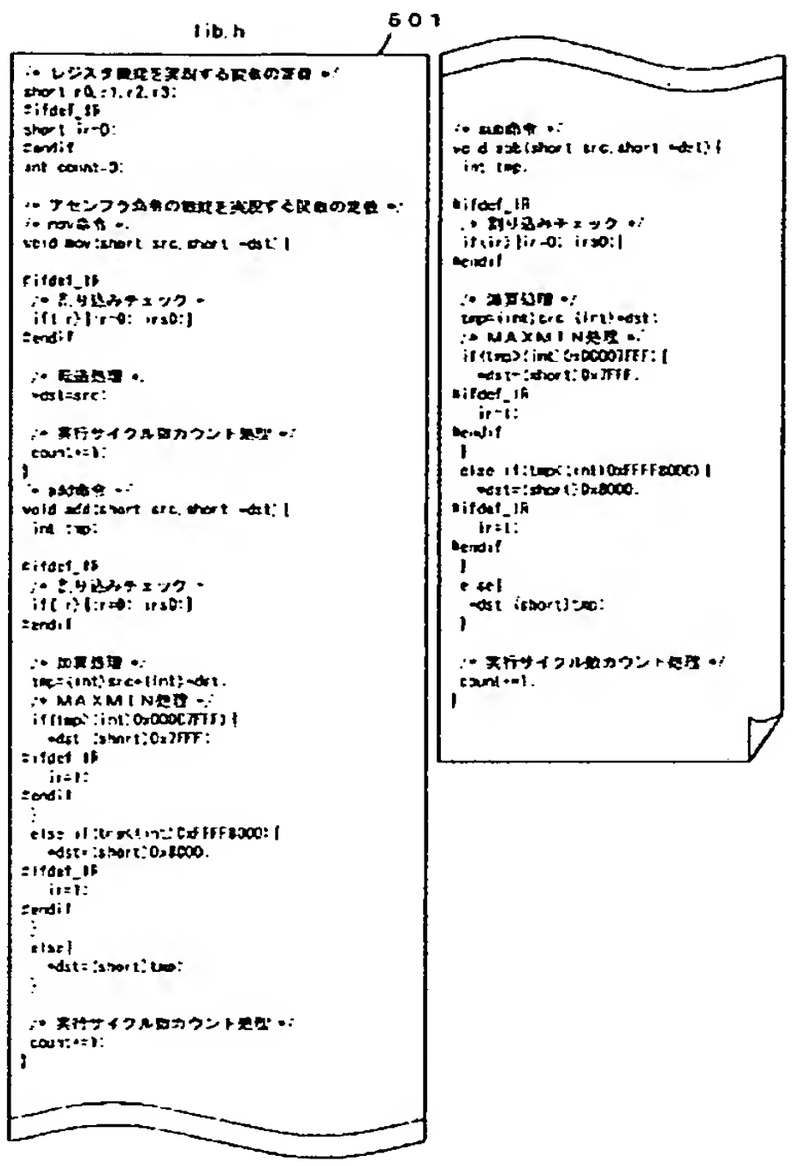

【図６】



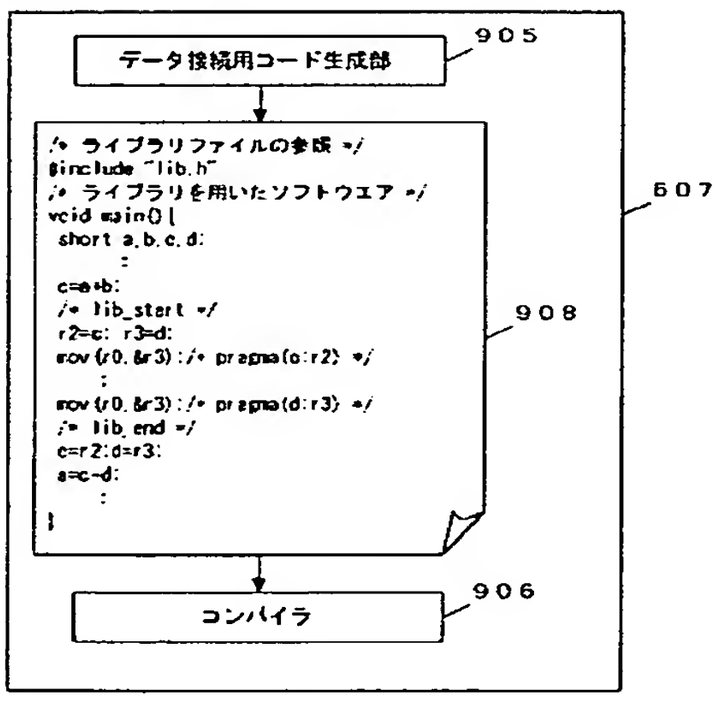
【図７】



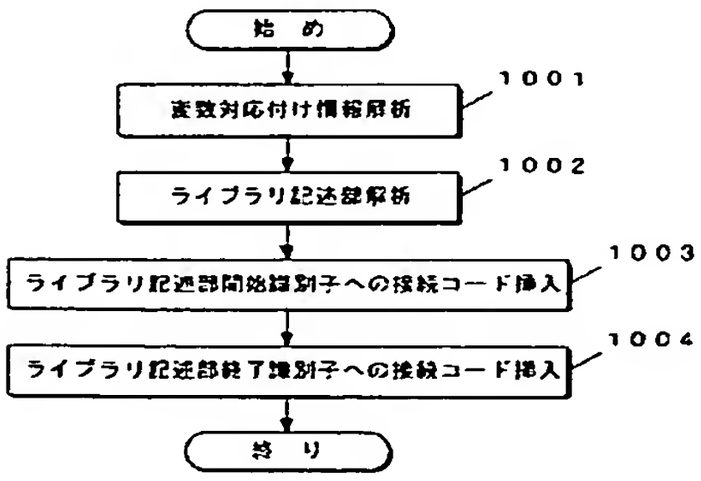

【図８】




【図９】




【図１０】




【図１１】